Opturion Load Manager – Data Formats and Web Service

Version 18

25 June 2025

OPTURION

Intelligent Decision Support

Contents

0 Overview

The Opturion Load Manager (OLM) is a payload optimiser for fuel tankers. It takes a set of orders, which take the form of volumes of different types of fuel to be delivered to one or more sites, and a fuel tanker vehicle, and allocates (assigns) volumes of fuel to the compartments on the vehicle. The OLM generates load plans (profiles) that aim to deliver the required volumes, and if possible more, up to the point of a maximum payload, while complying with regulations on axle weights and Gross Combination Mass (GCM) limits. It enforces these limits on each transport leg in the case of split loads (deliveries at multiple sites).

Alternatively, the OLM can be used to check that a proposed load plan satisfies regulations on axle weights and GCM limits.

This document describes the web services used by the OLM, together with the input and output data formats that it uses.

The input and output to the OLM can be given either as a Microsoft Excel workbook (OLM-Excel, see sections 2 and 3) or as a JSON document (OLM-JSON, see sections 4 and 5).

The OLM-Excel format is intended for use as a standalone modelling interface.

The OLM-JSON interface is intended for use when interacting with the OLM through a web service.

1 Concepts and Data Model

This section describes the major concepts in the OLM data model.

1.1 Optimising and Checking

The OLM has two modes of operation. These are:

1. *Optimising*: in this mode, the optimiser will *create* an optimal load plan.
2. *Checking*: in this mode, the optimiser will *check* that a supplied load plan satisfies the axle weight, GCM and capacity constraints. It will report any violations of these constraints.

1.2 Scenarios

An input to the OLM is called a *scenario*. It is a representation of the load planning problem you wish to solve. It consists of a number of customer orders to one or more sites, and a description of the vehicle that will deliver those orders. When *checking*, a scenario will also contain an existing load plan.

The data is described in the following sections.

1.3 Solutions

A *solution* is a load plan created or checked by the OLM. It is an allocation (assignment) of volumes and orders to the compartments of the vehicle. When checking, the solution will contain a report of axle weight, GCM or capacity constraints violated by the supplied load plan. (This report is not present in plans created by the OLM because it cannot create plans that contain constraint violations.)

1.4 Identifiers

Entities, such as orders, sites, axle groups, weight schemes, and products must have a unique *identifier* (Id) that is used to refer to them.

Identifier names must contain at least one character and must not contain the semicolon (`;`) character. They are not case or "white space" sensitive.

For example, "`T1`", "`t1`" and "`T 1 `" will all be treated as identifying the same entity by the optimiser because they only differ in case or the amount of white space they have.

Even though they may be composed of digits, identifiers are not numeric. For example, the OLM will treat "`1`" and "`001`" as being distinct identifiers.

(Users of the OLM-Excel format should be aware that there are Excel-specific restrictions on the specification of identifiers; see Section 2.1 for details.)

All references to an entity are made using its identifier.

1.5 Weights, Volumes and Distances

Weights and volumes in a scenario are to be specified in kilograms (kg) and litres (L) respectively. Distances (which describe the relative position of equipment) can be specified in arbitrary units, provided the same units are used consistently throughout the scenario (e.g. metres, centimetres).

1.6 Orders

Orders are volumes of product to be delivered to a site. Orders are allocated to compartments on the vehicle. Each order will be allocated to one or more compartments (that is, its volume may be spread across multiple compartments), but no two orders will be allocated to the same compartment.

- **Site**
  Each order is to be delivered to a given site. Sites are visited in the order they are listed in the input (Section 1.7).
- **Product**
  Each order is for a given product (e.g. ULP, Diesel) (Section 1.8).
- **Volumes**
  Each order has a *target volume*, which we aim to deliver, and a *maximum volume*, which we potentially could deliver. The optimisation will attempt to allocate the target volumes as a first priority, and more volume (but not exceeding the maximum), if it can.

  An order *may* have a *minimum compartment volume*, which is a minimum volume of product that must be loaded into *each* compartment that is allocated to the order. The minimum compartment volume may be required to account for pump ramp-up / ramp-down times at terminals, or to facilitate the addition of additives after the order is loaded.

Example In the following example, we have four orders being delivered to two sites.

Id	Site	Product	Target Volume	Maximum Volume
Order 1	Site 1	ULP	11,750	15,000
Order 2	Site 1	Diesel	14,800	20,000
Order 3	Site 2	ULP	8,000	15,000
Order 4	Site 2	Diesel	9,000	15,000

Table 1: Orders example.

The first two orders are to `Site 1`, and consist of target volumes of 11,750 L of ULP and 14,800 L of Diesel (with a maximum of 15,000 L and 20,000 L respectively). The last two orders are to `Site 2`, and consist of target volumes of 8,000 L of ULP and 9,000 L of Diesel (with a maximum of 15,000 L in both cases).

Orders may be marked as *required*. A load plan created by the optimiser *must* include a non-zero quantity of each required order. Required orders must be used carefully, since they make it possible to create unsatisfiable scenarios where it is impossible to create a load plan.

1.7 Sites

Each delivery site has an associated set of weight restrictions. These weight restrictions are defined using weight schemes (Section 1.11). Sites are visited in the order they appear here. The weight restrictions apply to the leg towards visiting the site. For example, if we are delivering to two sites, then there are three legs:

1. from the terminal to the first site
2. from the first site to the second site
3. from the second site back to the depot/terminal

(It is assumed the last leg has no specific weight restrictions, as the vehicle is empty at that stage.) The weight restrictions for each site are determined from the roads traversed on the way to that site.

Example Building on the previous example, we define a weight scheme for each of the sites.

Id	Weight Scheme
Site 1	HML
Site 2	GML

Table 2: Sites example.

Here, we allow higher mass limit (HML) on the first leg and general mass limit (GML) on the second leg. The weight schemes themselves are defined in Section 1.11.

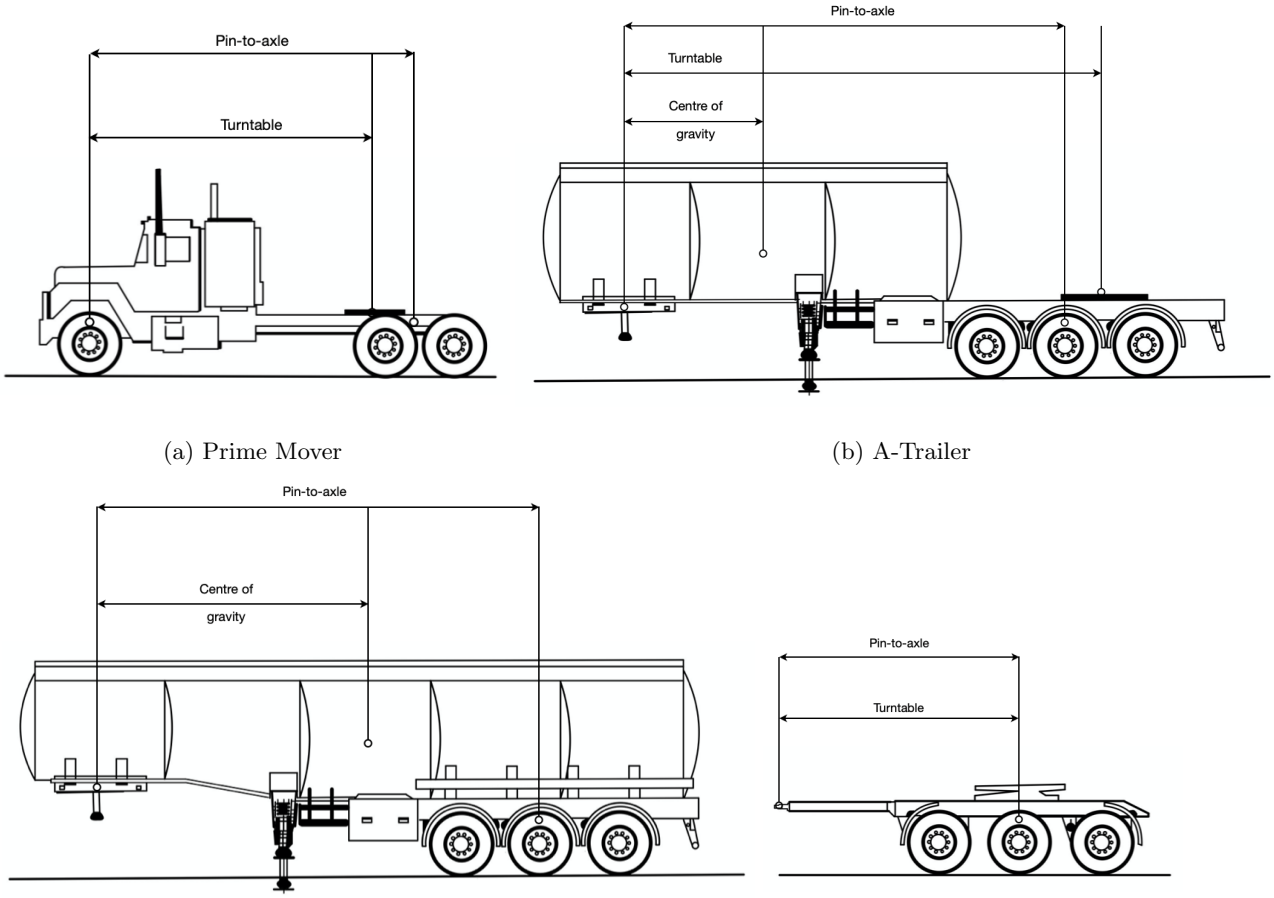
1.8 Products

Each product has a specified density (specific gravity) that determines the weight of a given volume of the product. The density is measured in kg/L.

Example An example defining the products referred to in the orders.

Id	Density
ULP	0.74
Diesel	0.85

Table 3: Products example.



(a) Prime Mover

(b) A-Trailer

(c) B-Trailer

(d) Dolly

Figure 1: Vehicle measurements.

1.9 Axle Groups

The vehicle is described in terms of its axle groups and its compartments. Each axle group corresponds to a semi-trailer or dolly. An axle group has one set of wheels, and hooks into the preceding axle group through either a fifth-wheel coupling (turntable) or a drawbar (in a dolly). A prime mover is modelled as if it were a dolly followed by a semi-trailer. Figure 1 shows an example of these.

For each axle group, we define:

- **Pin-to-axle**
  The distance between the kingpin and the centre of the axle group. In case of a dolly, this would notionally be the distance between the front of the drawbar and the centre of the axle group, but because the length of the drawbar has no material effect on the load build, we can use a notional value of, for example, one metre.

- **Has Turntable**
  Whether the axle group has a turntable?
- **Turntable**
  The position of the turntable (if applicable) as measured from the kingpin. If this is less than the pin-to-axle distance, then it means that the turntable is in front of the centre of the axle group, otherwise it is behind it.
- **Pin Tare**
  The tare weight of the trailer/dolly as measured under the kingpin.
- **Axle Tare**
  The tare weight of the trailer/dolly as measured under the axle group (i.e. under the wheels).

Example Axle groups for a B-double:

Id	Pin-to-axle	Has Turntable	Turntable	Pin Tare	Axle Tare
Steer	1	true	1	0	4,096
Drive	4,800	true	4,450	0	5,475
T1	7,950	true	8,250	1,650	4,450
T2	7,850	false	0	1,300	4,950

Table 4: Example axle groups for a B-double.

The first two axle groups are on the prime mover: `Steer` is the front wheels and `Drive` is the rear wheel set. The front of the prime mover weighs 4,096 kg and the rear weighs 5,475 kg. The distance between the front and rear wheels is 4,800 mm (we have chosen millimetres as the unit of distance here). The turntable on the prime mover is positioned 4,450 mm from the front wheels (i.e. it is 350 mm in front of the centre of the rear wheel set). Note that the front wheels are essentially a dolly. We set a notional pin-to-axle and turntable position right above the axle group.

The first trailer, `T1`, has a pin-to-axle distance of 7,950 mm and a turntable that is 300 mm behind the centre of the axle group (at 8,250 mm from the kingpin). The tare weights under the kingpin and wheels are 1,650 kg and 4,450 kg respectively.

The second trailer, `T2`, has a pin-to-axle distance of 7,850 mm and has no turntable. The tare weights under the kingpin and wheels are respectively 1,300 kg and 4,950 kg.

Example Axle groups for a rigid vehicle:

Id	Pin-to-axle	Has Turntable	Turntable	Pin Tare	Axle Tare
Front	1	true	1	0	7,590
Rear	5,087	false	0	0	5,470

Table 5: Example axle groups for a rigid vehicle.

We model rigid vehicles as a trailer hooked directly to the front axle group. The distance between the front and rear wheels is 5,087 mm. The tare weight under the front wheels is 7,590 kg and the tare weight under the rear wheels is 5,470 kg.

Example Axle groups for a rigid vehicle with a dog trailer:

Id	Pin-to-axle	Has Turntable	Turntable	Pin Tare	Axle Tare
Front	1	true	1	0	7,590
Rear	5,087	false	0	0	5,470
DogFront	1	true	1	0	2,660
DogRear	8,071	false	0	0	4,670

Table 6: Example axle groups for a rigid vehicle with a dog trailer

We model dog trailers as a trailer hooked directly to a notional dolly. This dolly corresponds to the front axle group of the trailer. The distance between the trailer's front and rear axle wheels is 8,071 mm. The tare weight under the trailer's front wheels is 2,660 kg and the tare weight under its rear wheels is 5,470 kg.

The OLM supports arbitrary vehicle configurations, from single-trailer vehicles through to road trains.

1.10 Compartments

The compartments on the trailers are defined by:

- **Trailer Axle Group**
  The axle group of the trailer the compartment belongs to.
- **Safe Fill Level**
  The maximum volume (in L) that can be safely transported in the compartment.
- **Capacity**
  The capacity of the compartment (in L). This is needed by the regulations on large compartments.
- **Centre of Gravity**
  The centre of gravity of the compartment, as measured from the kingpin of the trailer the compartment belongs to. For the first compartment of a trailer, the centre of gravity may be negative as the kingpin may be behind it.

Example An example of the compartments for the B-double in Table 4.

Id	Trailer Axle Group	Safe Fill Level	Capacity	Centre of Gravity
1	T1	8,340	8,600	100
2	T1	8,340	8,600	2,390
3	T1	8,340	8,600	4,690
4	T2	7,760	8,000	500
5	T2	7,760	8,000	2,490
6	T2	8,340	8,600	4,555
7	T2	8,340	8,600	6,695
8	T2	8,340	8,600	8,830

Table 7: Compartment example.

Here, there are eight compartments, three on the front trailer (`T1`) and five on the rear trailer (`T2`).

1.11 Weight Schemes

Each of the weight schemes as referred to in the Sites table, is defined in terms of maximum weights on each of the vehicle axle groups (as defined in Section 1.9) as well as on the vehicle combination (GCM: Gross Combination Mass).

- **Axle Group**: the axle group to which the weight limit applies, or **GCM** to define the Gross Combination Mass limit.
- **Limit**: the limit in kg.

Example For the weight schemes defined in Table 2 and axle groups of the B-double from Table 4, we can define weight limits as follows:

Id	Axle Group	Limit
GML	Steer	6,000
GML	Drive	16,500
GML	T1	20,000
GML	T2	20,000
GML	GCM	62,500
HML	Steer	6,000
HML	Drive	17,000
HML	T1	22,500
HML	T2	22,500
HML	GCM	68,000

Table 8: Weight schemes example.

1.12 Allocations

In a solution, an order and a volume is allocated (assigned) to each compartment.

Example An example output for the vehicle whose compartments are given in Table 7.

Compartment	Order	Product	Volume
1	4	Diesel	8,340
2	4	Diesel	1,250
3	3	ULP	8,340
4	1	ULP	7,760
5	2	Diesel	3,320
6	2	Diesel	8,340
7	2	Diesel	8,340
8	1	ULP	7,240

Table 9: Allocations Example.

Here, the first three compartments (the front trailer) are used for the second site (orders 3 and 4) with a total delivered quantity of 8,340 L of ULP and 9,590 L of Diesel. The five compartments of the rear trailer are used for the first site and deliver a total of 15,000 L of ULP and 20,000 L of Diesel.

1.13 Weight Calculation

In this section, we describe how the optimiser calculates the weights on each axle group for a given vehicle configuration and allocation. Table 10 shows a sample allocation to the B-double from Table 4, whose compartments are shown in Table 7. We use this as a running example.

Trailer Axle Group	Compartment	Order	Product	Volume	Weight
T1	1	4	Diesel	8,340	7,089.0
T1	2	4	Diesel	1,250	1,062.5
T1	3	3	ULP	8,340	6,171.6
T2	4	1	ULP	7,760	5,742.4
T2	5	2	Diesel	3,320	2,822.0
T2	6	2	Diesel	8,340	7,089.0
T2	7	2	Diesel	8,340	7,089.0
T2	8	1	ULP	7,240	5,357.6

Table 10: Example allocation.

We begin by calculating the weight for each compartment. This is calculated using the following formula:

$$compartment\_weight(c) = compartment\_volume(c) \cdot density(p) \tag{1}$$

where c is a compartment, and p the product in that compartment. For example, the first compartment in Table 10 contains 8,340 L of Diesel. This will have compartment weight of $8{,}340\,\text{L} \cdot 0.85\,\text{kg/L} = 7{,}089\,\text{kg}$. The fifth column of Table 10 shows the calculated weights for each compartment in the example.

Next, for each axle group, we calculate a *pin weight* and an *axle weight*. For axle groups that do *not* include a turntable, the formulas to do this are:

$$\begin{aligned} pin\_weight(ag) =& pin\_tare(ag) \\ &+ \sum_{c \in compartments(ag)} compartment\_weight(c) \cdot \frac{pin\_to\_axle(ag) - CoG(c)}{pin\_to\_axle(ag)} \end{aligned} \tag{2}$$

$$\begin{aligned} axle\_weight(ag) =& axle\_tare(ag) \\ &+ \sum_{c \in compartments(ag)} compartment\_weight(c) \cdot \frac{CoG(c)}{pin\_to\_axle(ag)} \end{aligned} \tag{3}$$

That is, the pin/axle weight is the pin/axle tare, plus a proportion of the compartment weights on the axle group based on how far the centre of gravity of the compartment is from the pin/axle. For example, considering the rear trailer (`T2`) of the B-double and allocations given from Table 10. The pin weight equals

$$\begin{aligned} pin\_weight(T2) =& 1,300 \\ &+ 5,742.4 \cdot \frac{7,850 - 500}{7,850} \\ &+ 2,822.0 \cdot \frac{7,850 - 2,490}{7,850} \\ &+ 7,089.0 \cdot \frac{7,850 - 4,555}{7,850} \\ &+ 7,089.0 \cdot \frac{7,850 - 6,695}{7,850} \\ &+ 5,357.6 \cdot \frac{7,850 - 8,830}{7,850} \\ =& 11,953.3 \end{aligned} \tag{4}$$

and the axle weight equals

$$\begin{aligned} axle\_weight(T2) =& 4,950 \\ &+ 5,742.4 \cdot \frac{500}{7,850} \\ &+ 2,822.0 \cdot \frac{2,490}{7,850} \\ &+ 7,089.0 \cdot \frac{4,555}{7,850} \\ &+ 7,089.0 \cdot \frac{6,695}{7,850} \\ &+ 5,357.6 \cdot \frac{8,830}{7,850} \\ =& 22,396.7 \end{aligned} \tag{5}$$

For axle groups that do have a turntable, we also need to add the weight of the next axle group's pin weight to the formula:

$$\begin{aligned} pin\_weight(ag) =& pin\_tare(ag) \\ &+ \sum_{c \in compartments(ag)} compartment\_weight(c) \cdot \frac{pin\_to\_axle(ag) - CoG(c)}{pin\_to\_axle(ag)} \\ &+ pin\_weight(ag+1) \cdot \frac{pin\_to\_axle(ag) - turntable(ag)}{pin\_to\_axle(ag)} \end{aligned} \tag{6}$$

$$\begin{aligned} axle\_weight(ag) =& axle\_tare(ag) \\ &+ \sum_{c \in compartments(ag)} compartment\_weight(c) \cdot \frac{CoG(c)}{pin\_to\_axle(ag)} \\ &+ pin\_weight(ag+1) \cdot \frac{turntable(ag)}{pin\_to\_axle(ag)} \end{aligned} \tag{7}$$

For example, for the front trailer (`T1`) of the B-double and allocations as before, the pin weight equals

$$\begin{aligned} pin\_weight(T1) =& 1,650 \\ &+ 7,089.0 \cdot \frac{7,950 - 100}{7,950} \\ &+ 1,062.5 \cdot \frac{7,950 - 2,390}{7,950} \\ &+ 6.171.6 \cdot \frac{7,950 - 4,690}{7,950} \\ &+ 11,953.3 \cdot \frac{7,950 - 8,250}{7,950} \\ =& 11,472.6 \end{aligned} \tag{8}$$

and the axle weight equals

$$\begin{aligned} axle\_weight(T1) =& 4,450 \\ &+ 7,089.0 \cdot \frac{100}{7,950} \\ &+ 1,062.5 \cdot \frac{2,390}{7,950} \\ &+ 6.171.6 \cdot \frac{4,690}{7,950} \\ &+ 11,953.3 \cdot \frac{8,250}{7,950} \\ =& 20,903.8 \end{aligned} \tag{9}$$

The drive and steer axle groups on the prime mover, as well as any dollies, function similarly, except that they do not have any compartments.

$$pin\_weight(Drive) = 0 + 11,472.6 \cdot \frac{4,800 - 4,450}{4,800} = 836.5 \tag{10}$$

$$axle\_weight(Drive) = 5,475 + 11,472.6 \cdot \frac{4,450}{4,800} = 16,111.1 \tag{11}$$

$$pin\_weight(Steer) = 0 + 836.5 \cdot \frac{1 - 1}{1} = 0.0 \tag{12}$$

$$axle\_weight(Steer) = 4,096 + 836.5 \cdot \frac{1}{1} = 4,932.5 \tag{13}$$

Table 11 lists of all the axle weights as we have calculated them, together with the GCM. The GCM is the sum of all of the axle weights.

Axle Group	**Weight**
Steer	4,932.5
Drive	16,111.1
T1	20,903.8
T2	22,396.7
GCM	64,344.1

Table 11: Axle Weights Example.

1.14 Stability Constraints

The optimiser enforces certain constraints to avoid vehicle instability due to liquid sloshing.

- If any compartment on the vehicle is used (i.e., non-empty), then the first compartment needs to be filled up to a user-specified minimum percentage of its safe fill level.
- If any compartment of a particular trailer is used apart from the first compartment on the first trailer, then either the last compartment of this trailer or the first compartment of the next trailer, needs to be filled up to a user-specified minimum percentage of its safe fill level.

As an example, take the B-double specified in Table 4 and a user-specified minimum percentage of the safe fill level of 100%. If any compartment is used, then compartment 1 must be full (up to its safe fill level of 8,340 L). If compartments 2 or 3 are used, then either compartment 3 or compartment 4 must be full. Finally if any of compartments 4 to 8 is used, then compartment 8 must be full.

1.15 Large Compartment Ullage Restrictions

The optimiser optionally enforces a constraint to restrict the amount of ullage in large compartments when the vehicle is loaded with certain products. The requirements of this constraint are taken from section `10.3.1.2` of the "Australian Code for the Transport of Dangerous Goods by Road & Rail". [1] The code requires that compartments with a capacity greater than 8,600 L not be loaded if the ullage in the compartment is more than 20% but less than 85%. (Or equivalently if the compartment is more than 15% but less than 80% full).

This constraint is enabled by default; it can be turned off if not applicable.

[1] https://www.ntc.gov.au/codes-and-guidelines/australian-dangerous-goods-code

1.16 Forbidden Compartments

The optimiser can be configured to prevent specific orders from being allocated to specific compartments. We refer to such compartments as being *forbidden* for those orders.

Using this mechanism, you can also prevent specific *products* from being allocated to a specific compartment (i.e. by forbidding all orders containing that product from the compartment).

1.17 Additional Fill Level Limits

The optimiser can be configured to enforce additional order- and compartment-specific minimum and maximum fill level limits. Operating permits sometimes impose such fill level constraints.

Additional fill level limits must satisfy the following conditions:

1. A fill level limit cannot exceed a compartment's safe fill level.
2. If both a minimum and maximum fill level limit are given for a compartment and order, then the minimum fill level limit must not exceed the maximum fill level limit.

2 Microsoft Excel Input

This section describes the Microsoft Excel input format for the optimiser.

The Excel input must be an Office Open XML workbook (`.xlsx`) or macro-enabled workbook (`.xlsx`). Office Open XML binary workbooks (`.xlsb`) are *not* currently supported. Workbooks in the old Excel binary file format (`.xls`) are *not* supported.

The workbook should contain the sheets described in the following sections. Those sheets that are specified as "required" must be present in the workbook. The workbook can contain sheets other than those recognised by the OLM-Excel format. Such extra sheets are ignored by the OLM.

Unless otherwise noted, sheets recognised by the optimiser may contain columns other than those required by the OLM-Excel format. Such extra columns are ignored by the OLM.

Columns recognised by the OLM may appear in any order.

All sheets must have a non-empty first row.

The OLM will evaluate any formulas used in the workbook when it is loaded. Formulas must *not* refer to sheets in other workbooks.

Not all Excel functions are supported by the OLM. All of the unsupported functions are listed in Appendix A.

2.1 Identifiers in Excel

Identifiers in OLM-Excel may be provided either as either text or numeric data. Identifiers given in text cells are taken *as-is* subject only to the rules concerning case and "white space" insensitivity. Identifiers given in numeric cells are *only* accepted by the optimiser if they are integer (i.e. their fractional part is zero). Formulas that evaluate to either text or (integer) numeric data may also be used to specify identifiers in Excel.

2.2 Sheet: Orders

This sheet is required and must contain at least one order.

2.2.1 Column: Id

Required: Yes.
Type: Number (integer).
Description: A unique identifier for the order.

2.2.2 Column: Site

Required: Yes.
Type: Text or (integer) number.
Description: A site identifier from the Sites sheet: the site to which the order is to be delivered.

2.2.3 Column: Product

Required: Yes.
Type: Text or (integer) number.
Description: A product identifier from the Products sheet: the product being delivered.

2.2.4 Column: Minimum Compartment Volume

Required: No.
Type: Number (nonnegative integer).
Description: A minimum volume that must be loaded into each compartment that this order is assigned to.
Default: `0`.

2.2.5 Column: Target Volume

Required: Yes.
Type: Number (integer).
Description: The target volume to be delivered for this order; must be greater than zero.

2.2.6 Column: Maximum Volume

Required: Yes.
Type: Number (integer).
Description: The maximum volume to be delivered for this order; must be greater than the target volume.

2.2.7 Column: Required

Required: No.
Value Type: Text (Boolean).
Description: If `TRUE`, then this order is required.
Default: `FALSE`.

2.3 Sheet: Sites

This sheet is required. It must contain an entry for each site that is referred to on the Orders sheet. Sites are visited in the order in which they are listed on this sheet.

2.3.1 Column: Id

Required: Yes.
Type: Text or (integer) number.
Description: A unique identifier for the site.

2.3.2 Column: Weight Scheme

Required: Yes.
Type: Text or (integer) number.
Description: A weight scheme identifier from the Weight Schemes sheet. The weight scheme that applies to the leg towards this site.

2.4 Sheet: Products

This sheet is required. It must contain an entry for each product that is referred to on the Orders sheet.

2.4.1 Column: Id

Required: Yes.
Type: Text or (integer) number.
Description: A unique identifier for the product.

2.4.2 Column: Density

Required: Yes.
Type: Number.
Description: The relative density of the product in kg/L. Must be nonnegative.

2.5 Sheet: Axle Groups

This sheet is required. It must contain an entry for each axle group on the vehicle. Axle groups must be listed in order, from front to rear.

2.5.1 Column: Id

Required: Yes.
Type: Text or (integer) number.
Description: A unique identifier for the axle group. The identifier must *not* be **GCM** as this is a reserved to define the Gross Combination Mass limit in weight schemes.

2.5.2 Column: Axle Count

Required: No.
Type: An (integer) number between `0` and `4` (inclusive).
Description: The number of axles in this axle group. A blank cell is equivalent to zero.

2.5.3 Column: Pin-to-Axle

Required: Yes.
Type: Number.
Description: The distance between the kingpin or front of the drawbar, to the centre of the axle group.

2.5.4 Column: Turntable

Required: Yes.
Type: Number.
Description: The distance between the kingpin or front of the drawbar, to the turntable. This cell can be left blank if there is no turntable on the axle group (i.e. on the trailer/dolly).

2.5.5 Column: Pin Tare

Required: Yes.
Type: Number.
Description: The tare weight as measured under the kingpin. Can be left blank (i.e. zero).

2.5.6 Column: Axle Tare

Required: Yes.
Type: Number.
Description: The tare weight as measured under the axle group (wheel set).

2.6 Sheet: Compartments

This sheet is required. It contains an entry for each compartment on the vehicle.

2.6.1 Column: Id

Required: Yes.
Type: Text or (integer) number.
Description: Each compartment must have a unique identifier.

2.6.2 Column: Trailer Axle Group

Required: Yes.
Type: Text or (integer) number.
Description: The axle group of the trailer to which this compartment belongs, as defined on the Axle Groups sheet.

2.6.3 Column: Safe Fill Level

Required: Yes.
Type: Number.
Description: The safe fill level (in litres) for the compartment.

2.6.4 Column: Capacity

Required: Yes.
Type: Number.
Description: The capacity (in litres) for the compartment.

2.6.5 Column: Centre of Gravity

Required: Yes.
Type: Number.
Description: The centre of gravity for the compartment, as measured from the kingpin.

2.7 Sheet: Weight Schemes

This sheet is required. It contains an entry for each weight scheme referred to on the Sites sheet and for each axle group defined on the Axle Groups sheet, as well as for the GCM (Gross Combination Mass). It has the following columns:

2.7.1 Column: Id

Required: Yes.
Type: Text or (integer) number.
Description: A unique identifier for the weight scheme.

2.7.2 Column: Axle Group

Required: Yes.
Type: Text or (integer) number.
Description: The identifier of the axle group to which the weight limit applies, as defined on the Axle Groups sheet, or the value **GCM** (for the Gross Combination Mass limit). Each axle group must have a weight limit.

2.7.3 Column: Limit

Required: Yes.
Type: Number.
Description: The weight limit (in kg) on the axle group or GCM.

2.8 Sheet: Forbidden Compartment Orders

This sheet is not required.

Each row defines a forbidden compartment rule preventing an order from being assigned to a compartment.

It has the following columns:

2.8.1 Column: Order

Required: Yes.
Type: Text or (integer) number.
Description: The unique identifier of an order from the orders sheet.

2.8.2 Column: Compartment

Required: Yes.
Type: Text or (integer) number.
Description: The unique identifier of an order from the compartments sheet.

3 Microsoft Excel Output

This description describes the Microsoft Excel output format for the optimiser.

The Excel output will contain references (e.g. site and order identifiers) to the scenario from which it was generated.

The optimiser generates output workbooks in Office OpenXML format (`.xlsx`).

3.1 Sheet: Allocations

This sheet contains assignments of (parts of) orders to a compartments. It has the following columns:

3.1.1 Compartment

Type: Text or (integer) number.
Description: Identifier of the compartment as defined on the Compartments sheet.

3.1.2 Order

Type: Text or (integer) number.
Description: Identifier of the order as defined on the Orders sheet.

3.1.3 Product

Type: Text or (integer) number.
Description: Identifier of the product as defined on the Products sheet. Note that the product is implied by the order.

3.1.4 Volume

Type: Number.
Description: The volume allocated to the compartment.

3.1.5 Weight

Type: Number.
Description: The weight allocated to the compartment.

4 JSON Input

This section describes the JSON input format to the optimiser.

Input to the OLM is given as a UTF-8 encoded JSON document containing either a single optimise request object or a single check request object.

Any fields that are omitted in the input take default values. Unless noted otherwise, these default values are `0` for numeric fields, `false` for Boolean fields and `null` for other types of field (including strings).

The structure of the JSON input is described in the following sections.

4.1 Optimise Request Object

The JSON input to the OLM when optimising is an **optimise request** object. It has the following fields:

1. `id`: a non-empty string giving a UUID (GUID) in canonical textual representation that uniquely identifies *each* invocation of the OLM by the client system. This field may be `null`.
2. `sites`: a non-empty array of site identifiers (strings). Sites are visited in the order they are listed in this array. It is an error for this field to be `null`.
3. `orders`: a non-empty array of order objects. It is an error for this field to be `null`.
4. `vehicle`: a vehicle object. It is an error for this field to be `null`.
5. `forbidden_compartment_orders`: an array of compartment-orders objects specifying any compartment and order assignments that are disallowed. This field may be `null`.
6. `fill_level_lower_limits`: an array of fill level objects specifying minimum fill level limits for specific compartments and orders. This field may be `null`.
7. `fill_level_upper_limits`: an array of fill level objects specifying maximum fill level limits for specific compartments and orders. This field may be `null`.
8. `timeLimit`: an integer, giving the number of seconds after which the optimisation is cut off and the best solution found so far is returned. A value of zero or less means that there is no time limit.

4.2 Check Request Object

The JSON input to the OLM when checking is a **check request** object. It contains all the fields of an optimise request object, plus the following additional field:

1. `assignments`: a non-empty array of check assignment objects.

It is an error for the `assignments` field to be `null`.

4.3 Order Object

An order is described by an object with the following fields:

1. `id`: a string giving the unique identifier for this order.
2. `tags`: an array of strings, giving client-specific information about this order. This field is *not* used by the optimiser. It may be `null` or empty.
3. `product`: a string giving the identifier for the product being delivered.
4. `density`: a number giving the relative density of the product in kg/L. Must be nonnegative.
5. `minimum_compartment_volume`: a number giving the minimum volume that must be loaded into *each* compartment that is assigned to this order.
6. `target_volume`: a number giving the target volume to be delivered for this order; must be greater than zero.
7. `maximum_volume`: a number giving the maximum volume that can be delivered for this order; must be greater than `target_volume`.
8. `site`: a string giving the identifier of the site to which this order must be delivered.
9. `required` A Boolean, if `true` then this order is required.

4.4 Vehicle Object

The vehicle is described by an object with the following fields:

1. `tags`: an array of strings, giving client-specific information about the vehicle. This field is *not* used by the optimiser. It may be `null` or empty.
2. `bookend_minimum_fill_percentage`: a number, between `0` and `1.0` (inclusive), that specifies the percentage of the safe fill level of a bookend compartment that is the minimum to which the compartment must be filled to satisfy the stability constraints. The default is `1` (i.e. `100%`).
3. `disable_large_compartment_ullage_restrictions`: a Boolean, if `true` then ullage restrictions for large compartments will *not* be applied. The default is `false` (i.e. ullage restrictions *will* be applied).
4. `axle_groups`: a non-empty array of axle group objects.
5. `gcm_limits`: a non-empty array of axle limit objects.
6. `compartments`: a non-empty array of compartment objects.

4.5 Axle Group Object

An axle group is described by an object with the following fields:

1. `id`: a string giving the unique identifier for this axle group.
2. `axle_count`: a integer between `0` and `4` (inclusive) give the number of axles in this axle group. Should be set to zero if the number of axles is not known.
3. `pin_to_axle`: a number giving the distance between the kingpin or front of the drawbar, to the centre of the axle group.
4. `has_turntable`: a Boolean, if `true` then there is a turntable on this axle group.
5. `turntable`: a number giving the distance between the kingpin or front of the drawbar, to the turntable; ignored if `has_turntable` is `false`.
6. `pin_tare`: a number giving the tare weight as measured under the kingpin.
7. `axle_tare`: a number giving the tare weight as measured under the axle group (wheel set).
8. `axle_limit`: an array of axle limit objects.

4.6 Axle Limit Object

An **axle limit** object describes a site specific weight limit for an axle group.

1. `site`: a string giving the identifier of the site to which this limit applies.
2. `limit`: a number giving the weight limit in kg.

4.7 Compartment Object

A compartment is described by an object with the following fields.

1. `id`: a string that defines the unique identifier for this compartment.
2. `tags`: an array of strings, giving client-specific information about this compartment. This field is *not* used by the optimiser. It may be `null` or empty.
3. `safe_fill_level`: an integer giving the maximum volume (in litres) that can be safely transported in the compartment.
4. `capacity`: an integer giving the maximum capacity of the compartment (in litres).
5. `centre_of_gravity`: a number giving the centre of gravity of the compartment, as measured from the kingpin of the trailer the compartment belongs to.
6. `trailer_axle_group`: the identifier of the axle group of the trailer the compartment belongs to.

4.8 Compartment-Orders Object

A **compartment-orders** object associates a set of orders with a compartment. It has the following fields:

1. `compartment`: a string giving the identifier of a compartment.
2. `orders`: an array of order identifiers.

The meaning of the association depends on the context in which the object is used.

4.9 Fill Level Object

A **fill level** object associates an order and compartment with a fill level. It has the following fields:

1. `order`: a string giving the identifier of an order.
2. `compartment`: a string giving the identified of compartment.
3. `level` an integer, give a fill level (in litres). (What this level means is context-dependent).

4.10 Check Assignment Object

A **check assignment** object describes the allocation of (part of) an order and volume to a specific compartment when using the OLM for checking. It has the following fields:

1. `order`: a string giving the identifier of the order that is allocated to the compartment.
2. `compartment`: a string giving the identifier of the compartment to which this (portion of the) order has been allocated.
3. `volume`: an integer, giving the volume (in litres) of the order that has been allocated to this compartment.

5 JSON Output

This section describes the JSON output generated by the optimiser.

5.1 Response Object

1. `id`: the UUID (GUID) from the `id` field of the request object that this is a response to. Will be `null` if the `id` field of the corresponding **request** object was `null`.
2. `status`: a string containing one of the response status codes.
3. `solution`: if this `status` field is `OK`, this is solution object giving the load plan; it is `null` otherwise.
4. `errors`: if `status` is `ERROR` then this field may contain an array of error strings; if `status` is any other value then this field will be `null`.
5. `warnings`: this field may contain an array of error strings that indicate probable problems with the input data; if there are no warnings then this field will be `null`.
6. `version`: a number giving the API version that was used to generate this response.
7. `build`: a string identifying what version of the OLM was used to generate this response.

5.2 Response Status Code

A **response status code** is used by the OLM to indicate if it succeeded or not. There are four possible values:

1. `OK`: evaluation was successful and a load plan was generated or checked.
2. `UNSATISFIABLE`: it is impossible to generate a load plan that satisfies the constraints given in the input. This response status is only returned when optimising.
3. `UNKNOWN`: evaluation timed out before a load plan could be generated. This response status is only returned when optimising.
4. `ERROR`: one or more errors occurred.

5.3 Error Strings

An **error string** is a string that describes some error or warning condition from the OLM. References to specific parts of the input or output data formats in error messages will be given using JSON Pointer notation. [2]

[2] https://tools.ietf.org/html/rfc6901

5.4 Solution Object

A **solution object** contains a load plan, together with KPIs for that load plan. When checking it also contains a report of any constraint violations in the supplied load plan.

1. `assignments`: an array of assignment objects that gives the allocation of orders and volumes to compartments.
2. `delivery_kpi`: an array of delivery KPI objects describing how the generated load plan meets the order volume requirements.
3. `axle_weights`: an array of axle weight objects describing the axle weights at each site.
4. `gcm_weights`: an array of gcm weight objects describing the GCM at each site.
5. `errors`: an array of strings, each of which is a report of a constraint violated by the assignments.
6. `structured_errors`: an array of solution error objects, each of which is a report of a constraint violated by the assignments.

5.5 Assignment Object

An **assignment** object describes the allocation of an order and volume to a specific compartment. It has the following fields:

1. `order`: a string giving the identifier of the order that is being allocated to a compartment.
2. `site`: a string giving the identifier of the site to which the order will be delivered.
3. `product`: a string giving the product.
4. `compartment`: a string giving the identifier of the compartment to which this (portion of the) order has been allocated.
5. `volume`: an integer giving the volume (in L) of the order that has been allocated to this compartment.
6. `weight`: a number giving the weight (in kg) of the order that has been allocated to this compartment.

5.6 Delivery KPI Object

A **delivery KPI** object contains KPIs for each order. It has the following fields.

1. `order`: a string giving the identifier of the order.
2. `site`: a string giving the identifier of the site to which the order will be delivered.
3. `product`: a string giving the product.
4. `target_volume`: a number giving the target volume to be delivered for this order.
5. `maximum_volume`: a number giving the maximum volume that can be delivered for this order.
6. `delivered_volume`: a number giving the volume of product that is scheduled to be delivered according to the generated load plan.

5.7 Axle Weight Object

An **axle weight** object contains the weight over an axle group at a given site for the load plan. It has the following fields:

1. `axle_group`: a string giving the identifier of the axle group for which the axle weight is reported.
2. `site`: a string giving the identifier of the site at which the axle weight is reported.
3. `weight`: a number giving the weight at the axle group.

5.8 GCM Weight Object

A **GCM weight** object contains the GCM at a given site for the load plan. It has the following fields:

1. `site`: a string, giving the identifier of a site.
2. `limit`: a number, giving the GCM limit at this site.
3. `weight`: a number, giving the GCM of the vehicle at this site.

5.9 Solution Error Objects

A **solution error** object describes a constraint that has been violated by the assignments. Unlike error strings, this representation allows client systems to present solution errors in their preferred way.

All solution error objects contain a field named **code**. This value of this field is one of the solution error codes. It distinguishes different types of solution error from one another. Each kind of solution error object contains additional fields that are specific to the type of error they describe.

5.9.1 Solution Error Codes

These describe the type of solution error object. The possible values are:

1. COMPARTMENT_CAPACITY_EXCEEDED
2. AXLE_WEIGHT_LIMIT_EXCEEDED
3. GCM_LIMIT_EXCEEDED
4. COMPARTMENT_FILL_BELOW_ORDER_MINIMUM
5. COMPARTMENT_SAFE_FILL_EXCEEDED
6. ORDER_MAXIMUM_VOLUME_EXCEEDED
7. FIRST_COMPARTMENT_FILL_INSUFFICIENT
8. FIRST_COMPARTMENT_EMPTY
9. REQUIRED_ORDER_NOT_ASSIGNED
10. LARGE_COMPARTMENT_ULLAGE_VIOLATION
11. FORBIDDEN_COMPARTMENT_FOR_ORDER
12. FILL_BELOW_MINIMUM_FILL_LEVEL
13. FILL_EXCEEDS_MAXIMUM_FILL_LEVEL

5.9.2 CompartmentCapacityExceeded Solution Error Object

A *CompartmentCapacityExceeded* solution error object describes the situation when the volume of product assigned to a compartment exceeds the maximum volume of the compartment. It has the following additional fields:

1. `order`: the identifier of the order assigned to the compartment.
2. `compartment`: the identifier of the compartment.
3. `capacity`: the maximum volume the compartment can hold.
4. `assigned_volume`: the volume assigned to the compartment.

Here is an example CompartmentCapacityExceeded solution error object:

```
{
 "code": "COMPARTMENT_CAPACITY_EXCEEDED,
 "order": "ORDER1",
 "compartment": "C3",
 "capacity": "1100",
 "assigned_volume": "1250"
}
```

5.9.3 AxleWeightLimitExceeded Solution Error Object

An *AxleWeightLimitExceeded* solution error object describes the situation where the weight over an axle group exceeds the weight limit for the axle group at a given site. It has the following additional fields:

1. `axle_group`: the axle group identifier.
2. `site`: the site identifier.
3. `weight`: the weight over the axle group.
4. `limit`: the weight limit for the axle group at the site.

Here is an example AxleWeightLimitExceeded solution error object:

```
{
 "code": "AXLE_WEIGHT_LIMIT_EXCEEDED",
 "axle_group": "AG2",
 "site": "SITE2",
 "weight": "1500",
 "limit": "1400"
}
```

5.9.4 GcmLimitExceeded Solution Error Object

A *GcmLimitExceeded* solution error object describes the situation where the GCM of a vehicle exceeds the GCM limit at a given site. It has the following additional fields:

1. `site`: the identifier of the site.
2. `limit`: the GCM limit at the site.
3. `gcm`: the GCM at the site.

Here is an example GcmLimitExceeded solution error object:

```
{
 "code": "GCM_LIMIT_EXCEEDED",
 "site": "SITE3",
 "limit": "13000",
 "gcm": "14500"
}
```

5.9.5 CompartmentFillBelowOrderMinimum Solution Error Object

A *CompartmentFillBelowOrderMinimum* solution error object describes the situation where the volume of an order assigned to a compartment is less than the minimum compartment volume for that order. It has the following additional fields:

1. `compartment`: the identifier of the compartment.
2. `order`: the identifier of the order.
3. `minimum_volume`: the minimum compartment volume for the order.
4. `assigned_volume`: the volume of the order assigned to this compartment.

Here is an example CompartmentFillBelowOrderMinimum solution error object:

```
{
 "code": "COMPARTMENT_FILL_BELOW_ORDER_MINIMUM",
 "compartment": "C1",
 "order": "ORDER2",
 "minimum_volume": "1000",
 "assigned_volume": "10"'
}
```

5.9.6 CompartmentSafeFillExceeded Solution Error Object

A *CompartmentSafeFillExceeded* solution error object describes the situation where a compartment has been filled beyond its safe fill level. It has the following additional fields:

1. `order`: the identifier of the order assigned to the compartment.
2. `compartment`: the identifier of the compartment.
3. `safe_fill_level`: the safe fill level of the compartment.
4. `assigned_volume`: the volume assigned to compartment.

Here is an example CompartmentSafeFillExceeded solution error object:

```
{
 "code": "COMPARTMENT_SAFE_FILL_EXCEEDED",
 "order": "ORDER4",
 "compartment": "C5",
 "safe_fill_level": "700,
 "assigned_volume": "900"
}
```

5.9.7 OrderMaximumVolumeExceeded Solution Error Object

A *OrderMaximumVolumeExceeded* solution error object describes the situation where the total volume of an order assigned to the vehicle exceeds the maximum volume for that order. It has the following additional fields:

1. `order`: the identifier of the order.
2. `site`: the identifier of the delivery site for the order.
3. `maximum_volume`: the maximum volume the can be delivered for this order.
4. `assigned_volume`: the volume of the order assigned to this vehicle.

Here is an example OrderMaximumVolumeExceeded solution error object:

```
{
 "code": "ORDER_MAXIMUM_VOLUME_EXCEEDED",
 "order": "ORDER4",
 "site": "SITE2",
 "maximum_volume": 6000,
 "assigned_volume": 8000
}
```

5.9.8 FirstCompartmentFillInsufficient Solution Error Object

A *FirstCompartmentFillInsufficient* solution error object describes the situation where the load in the first compartment is not sufficient to satisfy the vehicle stability constraints. It has the following additional fields:

1. `order`: the identifier of the order assigned to the first compartment.
2. `site`: the identifier of the site where the first compartment is not sufficiently loaded.
3. `first_compartment`: the identifier of the first compartment.
4. `minimum_volume`: the volume that must be assigned to the first compartment to satisfy the stability constraints.
5. `assigned_volume`: the volume that *is* assigned to the first compartment.

Here is an example FirstCompartmentFillInsufficient solution error object:

```
{
  "code": "FIRST_COMPARTMENT_FILL_INSUFFICIENT"
  "order": "ORDER3",
  "site": "SITE1",
  "first_compartment": "C1",
  "minimum_volume": 8340,
  "assigned_volume": 40,
},
```

5.9.9 FirstCompartmentEmpty Solution Error Object

A *FirstCompartmentEmpty* solution error object describes the situation where the first compartment needs to be filled to ensure vehicle stability, but it is empty. It has the following additional fields:

1. `site`: the identifier of the site where the first compartment is empty.
2. `first_compartment`: the identifier of the first compartment.
3. `non_empty_compartments`: an array of compartment identifiers that are non-empty at the given site.

Here is an example FirstCompartmentEmpty solution error object:

```
{
  "code": "FIRST_COMPARTMENT_EMPTY"
  "site": "SITE2",
  "first_compartment": "C1",
  "non_empty_compartments": [
    "C2", "C3"
  ]
}
```

5.9.10 BookendCompartmentFillInsufficient Solution Error Object

A *BookendCompartmentFillInsufficient* solution error object describes the situation where the load in one of the bookend compartments is not sufficient to satisfy the vehicle stability constraints. It has the following additional fields:

1. `order`: the identifier of the order assigned to the bookend compartment.
2. `site`: the identifier of the site where the bookend compartment is not sufficiently loaded.
3. `compartment`: the identifier of the bookend compartment.
4. `minimum_volume`: the volume that must be assigned to the bookend compartment to satisfy the stability constraints.
5. `assigned_volume`: the volume that *is* assigned to the bookend compartment.

Here is an example BookendCompartmentFillInsufficient solution error object:

```
{
  "code": "BOOKEND_COMPARTMENT_FILL_INSUFFICIENT"
  "order": "ORDER3",
  "site": "SITE1",
  "compartment": "C1",
  "minimum_volume": 8340,
  "assigned_volume": 40,
},
```

5.9.11 BookendCompartmentEmpty Solution Error Object

A *BookendCompartmentEmpty* solution error object describes the situation where at least one bookend compartment needs to be filled to ensure vehicle stability, but they are all empty. It has the following additional fields:

1. `site`: the identifier of the site where the compartment is empty.
2. `compartments`: the identifiers of the compartments.
3. `non_empty_compartments`: an array of compartment identifiers that are non-empty at the given site.

Here is an example BookendCompartmentEmpty solution error object:

```
{
  "code": "BOOKEND_COMPARTMENT_EMPTY"
  "site": "SITE2",
  "compartments": ["C1"],
  "non_empty_compartments": [
    "C2", "C3"
  ]
}
```

5.9.12 RequiredOrderNotAssigned Solution Error Object

A *RequiredOrderNotAssigned* solution error object describes the situation where an order is required to be assigned, but it is not. It has the following additional field:

1. `order`: the identifier of the order that is not assigned.

Here is an example RequiredOrderNotAssigned solution error object:

```
{
  "code": "REQUIRED_ORDER_NOT_ASSIGNED"
  "order": "ORDER4"
}
```

5.9.13 LargeCompartmentUllageViolation Solution Error Object

A *LargeCompartmentUllageViolation* solution error object describes the situation where the ullage restrictions on a large compartment have not been satisfied. It has the following additional fields:

1. `order`: the identifier of the order assigned to the compartment.
2. `compartment`: the identifier of the compartment that does not satisfy the ullage restrictions.
3. `assigned_volume`: the volume assigned to the compartment.

Here is an example LargeCompartmentUllageViolation solution error object:

```
{
    "code": "LARGE_COMPARTMENT_ULLAGE_VIOLATION",
"order": "ORDER3",
    "compartment": "C2"
    "assigned_volume: 6400
}
```

5.9.14 ForbiddenCompartmentForOrder Solution Error Object

A *ForbiddenCompartmentForOrder* solution error object describes the situation where the forbidden compartment constraint has been violated. It has the following additional fields:

1. `compartment`: the identifier of the compartment that does not satisfy the forbidden compartment constraint.
2. `order`: the identifier of the order that is assigned to the compartment.

Here is an example ForbiddenCompartmentForOrder solution error object:

```
{
   "code": "FORBIDDEN_COMPARTMENT_FOR_ORDER",
   "compartment": "C1",
   "order": "ORDER_4"
}
```

5.9.15 FillBelowMinimumFillLevel Solution Error Object

A *FillBelowMinimumFillLevel* solution error object describes the situation where the quantity of an order assigned to a compartment is below the level required by an additional fill level limit for that order and compartment. It has the following additional fields:

1. `order`: the identifier of the order that is assigned to the compartment and is the subject of the fill level limit.
2. `compartment`: the identifier of the compartment.
3. `assigned_volume`: the volume that *is* assigned to the compartment.
4. `minimum_fill_level`: the minimum fill required for the order and compartment by the additional fill level limit.

Here is an example FillBelowMinimumFillLevel solution error object:

```
{
    "code": "FILL_BELOW_MINIMUM_FILL_LEVEL",
    "order": "ORDER_4",
    "compartment": "C3",
    "assigned_volume": 2000,
    "minimum_fill_level": 6000
}
```

5.9.16 FillExceedsMaximumFillLevel Solution Error Object

A *FillExceedsMaximumFillLevel* solution error object describes the situation where the quantity of an order assigned to a compartment exceeds the level required by an additional fill level limit for that order and compartment. It has the following additional fields:

1. `order`: the identifier of the order that is assigned to the compartment and is the subject of the fill level limit.
2. `compartment`: the identifier of the compartment.
3. `assigned_volume`: the volume that *is* assigned to the compartment.
4. `maximum_fill_level`: the maximum fill required for the order and compartment by the additional fill level limit.

Here is an example FillExceedsMaximumFillLevel solution error object:

```
{
    "code": "FILL_EXCEEDS_MAXIMUM_FILL_LEVEL",
    "order": "ORDER_2",
    "compartment": "C1",
    "assigned_volume": 7200,
    "maximum_fill_level": 6000
}
```

6 Load Manager Web Interface

The OLM provides a simple web interface that allows OLM-Excel scenarios to be uploaded and evaluated. The resulting load plans can be downloaded as OLM-Excel output workbooks.

This interface is accessed by entering a URL of the following form into the navigation bar of your web browser:

`https://`*server-name*`/olm`

where *server-name* will be provided by Opturion.

7 Load Manager Web Service

7.1 Overview

This section describes the OLM web service. The web service lets clients send scenarios to the OLM over a HTTP/1.1 connection. Results are sent back to the client over the *same* HTTP connection.

Optimisation scenarios may be sent as either an OLM-JSON optimise request object or as an OLM-Excel workbook.

Checking scenarios can only be sent as an OLM-JSON check request object. (Excel input is not currently supported when checking.)

Optimisation results can be returned as either OLM-JSON or OLM-Excel as specified by the client.

Checking results can currently only be returned as OLM-JSON.

The web service has the following endpoints:

1. SOLVE-JSON – submit an OLM-JSON scenario to the OLM for load plan optimisation.
2. SOLVE-EXCEL – submit an OLM-Excel scenario to the OLM for load plan optimisation.
3. CHECK-JSON – submit an OLM-JSON scenario containing a load plan to the OLM for constraint checking.

Opturion will provide authentication credentials for use with the web service if required.

7.2 solve-json Endpoint

The SOLVE-JSON endpoint is used to submit scenarios to the optimiser in OLM-JSON format for load plan creation.

7.2.1 solve-json Requests

A SOLVE-JSON request is an HTTP POST request to a URL of the following form:

`https://`*server-name*`/olm/v1/solve-json?`*parameters*

where *server-name* will be provided by Opturion and *parameters* is described below.

The body of the POST request is a JSON document containing a OLM-JSON optimise request object.

7.2.2 solve-json Parameters

The SOLVE-JSON endpoint accepts one optional parameter. It is:

1. **format** – this parameter controls whether a successful output is in OLM-JSON or OLM-Excel format. Its value should be one of "`json`" or "`excel`". If this parameter is not present, then it defaults to "`json`".

7.2.3 solve-json Request Example

Here is an example SOLVE-JSON request URL:

```
https://demo.opturion.com/olm/v1/solve-json?format=excel
```

Note that in this case we are using the `format` parameter to make the OLM return the solution in OLM-Excel format.

7.2.4 solve-json Responses

The following table lists all of the HTTP status codes that SOLVE-JSON specifically returns and their meaning. Any other HTTP status code returned by SOLVE-JSON that is not covered by this table has its usual meaning.

HTTP Status Code	Description	HTTP Response Body
200 (OK)	A load plan has been returned.	A OLM-JSON response object or OLM-Excel output workbook.
400 (BAD REQUEST)	There are errors in the input or web service invocation.	The body of the HTTP response *may* contain an OLM-JSON response object describing any errors.
401 (UNAUTHORIZED)	Authentication (if required) failed; the client is not authorized to access the optimiser.	Empty.
429 (TOO MANY REQUESTS)	May be returned if rate limiting has been applied by the server.	Empty.
500 (INTERNAL SERVER ERROR)	Indicates that an error occurred on the server.	The body *may* contain a OLM-JSON response object further describing the error.

7.3 solve-excel Endpoint

The SOLVE-EXCEL endpoint is used to submit scenarios to the optimiser in OLM-Excel format for load plan creation.

7.3.1 solve-excel Requests

A SOLVE-EXCEL request is an HTTP POST request to a URL of the following form:

`https://`*server-name*`/olm/v1/solve-excel?`*parameters*

where *server-name* will be provided by Opturion and *parameters* is describe below.

The body of the POST request is an OLM-Excel workbook.

7.3.2 solve-excel Parameters

The SOLVE-EXCEL endpoint accepts one optional parameter. It is:

1. **format** – this parameter controls whether a successful out is in OLM-JSON or OLM-Excel format. Its value should be one of "`json`" or "`excel`". If this parameter is not present, then it defaults to "`excel`".

7.3.3 solve-excel Request Example

Here is an example SOLVE-EXCEL request URL:

```
https://demo.opturion.com/olm/v1/solve-excel?format=json
```

Note that in this case we are using the `format` parameter to make the OLM return the solution in OLM-JSON format.

7.3.4 solve-excel Responses

The following table lists all of the HTTP status codes that SOLVE-EXCEL specifically returns and their meaning. Any other HTTP status code returned by SOLVE-EXCEL that is not covered by this table has its usual meaning.

HTTP Status Code	Description	HTTP Response Body
200 (OK)	A load plan has been returned.	A OLM-JSON response object or OLM-Excel output workbook.
400 (BAD REQUEST)	There are errors in the input or web service invocation.	The body of the HTTP response *may* contain an OLM-JSON response object describing any errors.
401 (UNAUTHORIZED)	Authentication (if required) failed; the client is not authorized to access the optimiser.	Empty.
429 (TOO MANY REQUESTS)	May be returned if rate limiting has been applied by the server.	Empty.
500 (INTERNAL SERVER ERROR)	Indicates that an error occurred on the server.	The body *may* contain a OLM-JSON response object further describing the error.

7.4 check-json Endpoint

The CHECK-JSON endpoint is used to submit scenarios containing an existing load plan to the OLM in OLM-JSON format for load plan checking.

7.4.1 check-json Requests

A CHECK-JSON request is an HTTP POST request to a URL of the following form:

`https://`*server-name*`/olm/v1/check-json`

where *server-name* will be provided by Opturion.

The body of the POST request is a JSON document containing a OLM-JSON check request object.

7.4.2 check-json Parameters

The CHECK-JSON endpoint does not accept any optional parameters.

7.4.3 check-json Request Example

Here is an example CHECK-JSON request URL:

```
https://demo.opturion.com/olm/v1/check-json
```

7.4.4 check-json Responses

The following table lists all of the HTTP status codes that CHECK-JSON specifically returns and their meaning. Any other HTTP status code returned by CHECK-JSON that is not covered by this table has its usual meaning.

HTTP Status Code	Description	HTTP Response Body
200 (OK)	A checked load plan has been returned.	A OLM-JSON response object.
400 (BAD REQUEST)	There are errors in the input or web service invocation.	The body of the HTTP response *may* contain an OLM-JSON response object describing any errors.
401 (UNAUTHORIZED)	Authentication (if required) failed; the client is not authorized to access the optimiser.	Empty.
429 (TOO MANY REQUESTS)	May be returned if rate limiting has been applied by the server.	Empty.
500 (INTERNAL SERVER ERROR)	Indicates that an error occurred on the server.	The body *may* contain a OLM-JSON response object further describing the error.

7.5 Opturion Hosted Deployment

Opturion provides deployments of the OLM that are hosted by Amazon Web Services (AWS) in the Asia Pacific (Sydney) region.

Separate test and production deployments are provided.

7.5.1 Hosted Test Deployment

The details of the hosted test deployment are as follows.

The web service endpoint URLs are:

```
https://demo.opturion.com/olm/v1/solve-json
https://demo.opturion.com/olm/v1/solve-excel
https://demo.opturion.com/olm/v1/check-json
```

The web interface URL is:

```
https://demo.opturion.com/olm
```

7.5.2 Hosted Production Deployment

The details of the hosted production deployment are as follows.

The web service endpoint URLs are:

```
https://prod-au1.opturion.com/olm/v1/solve-json
https://prod-au1.opturion.com/olm/v1/solve-excel
https://prod-au1.opturion.com/olm/v1/check-json
```

The web interface URL is:

```
https://prod-au1.opturion.com/olm
```

7.5.3 Access to Opturion Servers

Opturion will provide access credentials to users of the OLM deployments hosted by Opturion.

All access to the Opturion servers must be made using HTTPS. HTTP connections will be automatically upgraded.

7.5.4 SSL/TLS Certificates

All Opturion servers use SSL/TLS certificates issued by the Let's Encrypt CA.

(Their "Certificate Compatibility" page [3] can be useful when diagnosing certificate compatibility problems in clients.)

[3] https://letsencrypt.org/docs/certificate-compatibility/

Appendices

A Unsupported Excel Functions

The following table lists all of the Excel functions that are *not* supported by the OLM.

ABSREF	DBCS	GET.CELL	LOGEST	RSQ
APP.TITLE	DDB	GET.DOCUMENT	LOGINV	SAVE.TOOLBAR
ARGUMENT	ENABLE.TOOL	GET.WINDOW	LOGNORMDIST	SEARCHB
ASC	END.IF	GET.WORKBOOK	MIDB	SKEW
BETADIST	ERROR	GET.WORKSPACE	N	SLN
BETAINV	EVALUATE	GETPIVOTDATA	NEGBINOMDIST	STEP
BINOMDIST	EXEC	GOTO	NUMBERSTRING	STEYX
CALL	EXPONDIST	GROWTH	PERMUT	SYD
CELL	FDIST	HARMEAN	PHONETIC	TINV
CHIDIST	FINDB	HYPGEOMDIST	PRESS.TOOL	TRIMMEAN
CHIINV	FINV	INFO	PROB	TTEST
CHITEST	FISHER	ISPMT	QUARTILE	TYPE
CONFIDENCE	FISHERINV	KURT	REGISTER.ID	USDOLLAR
CRITBINOM	FTEST	LAST.ERROR	RELREF	VDB
DATEDIF	GAMMADIST	LEFTB	REPLACEB	WEIBULL
DATESTRING	GAMMAINV	LENB	RETURN	WINDOW.TITLE
DB	GAMMALN	LINEST	RIGHTB	ZTEST